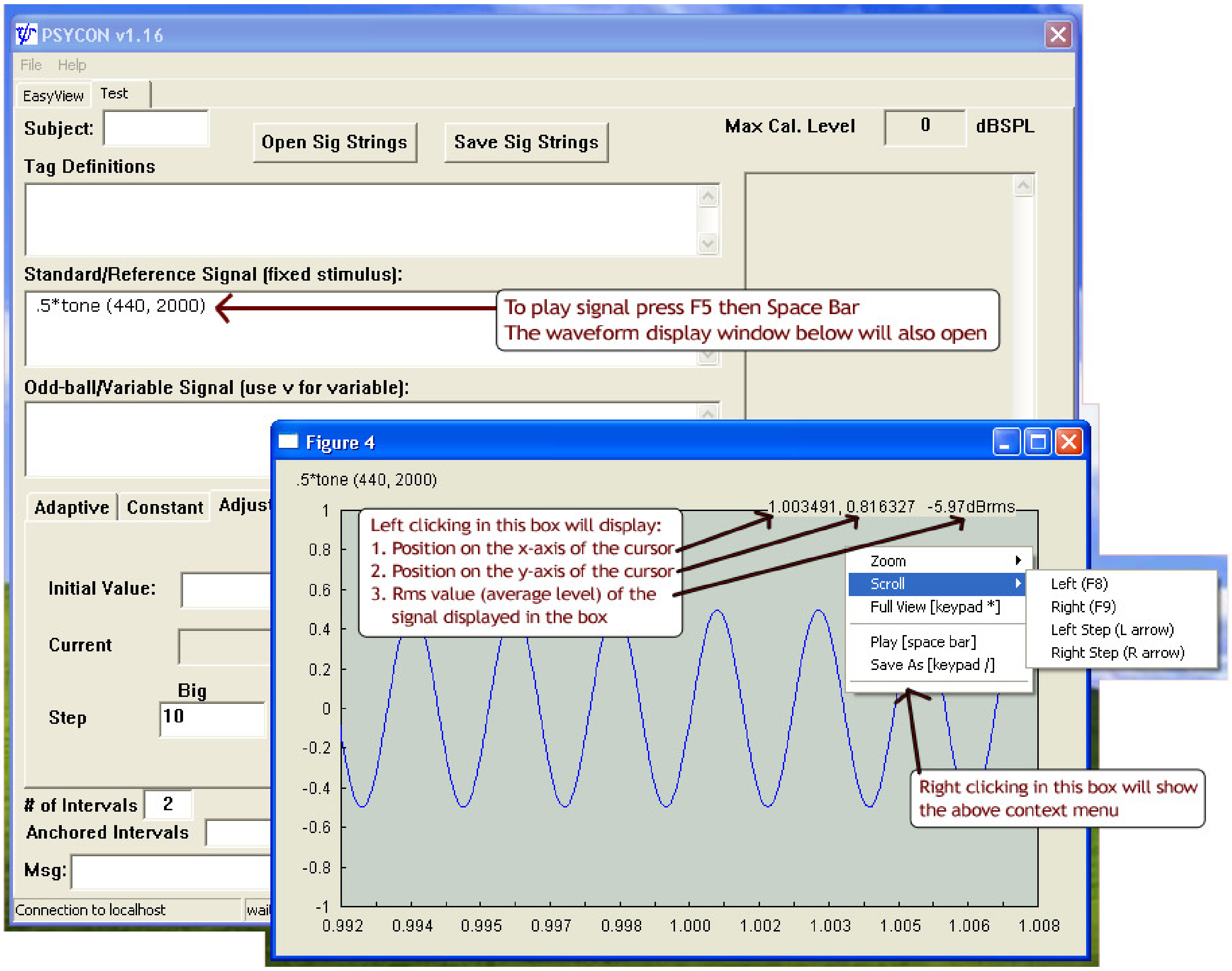
Psycon User's Manual

Developed by Bomjun Kwon
Manual partly written by
Jenny Shepherd (1st year AuD student, University of Utah)

Program Description: The purpose of the Psycon Software is to generate various kinds of signals to be used in tests and research.



.5*tone (440, 2000)

To play signal press F5 then Space Bar
The waveform display window below will also open

Left clicking in this box will display:
1. Position on the x-axis of the cursor
2. Position on the y-axis of the cursor
3. Rms value (average level) of the signal displayed in the box

Right clicking in this box will show the above context menu

1. Introduction

1.1. Purpose of psycon

To generate arbitrary sound stimuli for psychoacoustic research/education by intuitive and easy syntax

1.2. Structure of psycon program

The psycon software package comes with psycon_response. Psycon is for signal generation and psychoacoustic test administration and is to be used by the experimenter. Psycon_response is the program for the subject: it provides the instruction for the subject and collects responses from the subject. While both programs communicate with each other via the network protocol, the user only needs to specify the machine of the other program when launching each program to establish the connection (see).

1.3. Installation

Run psycon120.msi (this is version 1.20; it may be different for a different version) for psycon and psycon_resp_setup1.0.msi for psycon_response.

1.4. Signal generation and waveform display box Installation

Signals are defined by "psycon syntax" which is explained below, in signal definition edit boxes. With a signal typed in the signal edit box, press F5, then the signal will be plotted in the waveform display box, as seen above – you can bring up as many of these as signals you generate.

To play the signal: (click the waveform to play and) Press Spacebar

Keyboard Shortcuts while in the waveform display box

F2 - toggles signals of the left and right channels on the foreground of the display (**stereo signals only**)
F4 - displays the spectrum of the signal (to remove the spectrum view, press F4 again)
F8 - moves display of the signal to the left
F9 - moves display of the signal to the right
Left arrow - moves display of the signal to the left in smaller steps
Right arrow – moves display of the signal to the right in smaller steps
On number pad:
+ -zooms in on signal display
- -zooms out on signal display
***** -brings signal back to its original scale
/ -brings in "Save As" screen

Right click with mouse while in the signal box will show the above commands
Left click with mouse while in the signal box will show 3 numbers:

1st number – shows the position on the x-axis of the cursor
2nd number - shows the position on the y-axis of the cursor
3rd number – shows the rms (average level) of the signal within the signal box

2. Signal definition syntax

2.1. basic format in the syntax

Signals are represented in a "semi" mathematic style; i.e., you may add, subtract, multiple, or divide one signal or value with another, with a constraint of "common sense"—for example, you should not try to divide by zero. The most basic form of a signal is as follows:

```
scale_factor1 * signal
```

`signal` is either a function command (such as `tone` or `noise`) with a proper argument list, or a mathematic formula with an array of input argument specifying the time index (as if you were generating a signal in MATLAB). `scale_factor1` specifies the scaling of the signal; if it is skipped, it is presumed to be 1 (full or maximum scale). The simplest example of signal is as follows:

```
tone(500, 1000)
```

→ A pure tone with a frequency of 500-Hz and duration of 1000 milleseconds at the full scale.

```
0.25 * tone(500, 1000)
```

→ A pure tone with a frequency of 500-Hz and duration of 1000 milleseconds with the 25% magnitude from the full scale.

To add two signals, simply add them:

```
0.25 * tone(500, 1000) + 0.25 * tone(1000, 1000)
```

→ Two pure tones, each scaled by 25% of the maximum, with a duration of 1000 milliseconds are added: one is 500-Hz, the other is 1000-Hz.

If you followed thus far, you would know the problem of the following expression:

```
tone(500, 1000) + tone(1000, 1000)
```

That's right! The above expression tries to generate a signal that exceeds the full scale; as a result, the signal will be clipped and the desired signal will not be generated. Note that psycon does not issue an error message in this case, because this is technically not an error. The user always has to ensure that clipping does not occur.

If you want to use a dB factor for scaling, use the db function.

db	Calculate the actual factor from a dB value
Syntax	**db(db_value)**
Arguments	db_value: If positive, it calculates the scale factor with regard to the calibrated value. If negative, it calculates the scale factor of the specified value.
Examples	db(-6) Scale down by 6 dB db(-20) Scale down by 20 dB db(80) this specifies 20 dB down from the calibrated max level, which in this case is 100dB, → equivalent to db(-20)

Therefore, the following three expressions are equivalent:
0.5 * tone (300,500)
db(-6) * tone (300,500)
db(94) * tone (300,500) ← Assume that the cal. max level is 100dB

2.2. Time parameter (time marker)

Most functions that generate a signal require a time parameter enclosed by a bracket, which specifies the beginning and duration of the signal to generate. If the signal begins at the time 0, it can be skipped and only the duration is specified without a bracket, i.e.,

`tone(440,300)` is the same as `tone(440,[0 300])`.

If two tones occur in sequence, it would be as follows:

```
tone(440,[0 300]) + tone(880,[400 500])
```
→ a 440Hz-tone from 0 to 300 ms and another tone at 880Hz from 400 to 900 ms

3. Signal generation functions

tone	Generate a tone signal
Syntax	**`tone (freq, time_marker, [initial_phase=0])`**
Arguments	`freq:` frequency in Hz `time_marker:` [beginning_time duration] in msec. If only one value, it begins at 0
Examples	`tone (226, 400)` a 226-Hz tone from 0 to 400 ms `tone (226, [200 500])` a 226-Hz tone from 200 to 700 ms `tone (226, 400, .5)` ; 226-Hz tone of 400 ms, 180 degree out of phase

noise	Generate a white noise signal, uniformly distributed random val between -1 and 1
Syntax	**`noise (time_marker)`**
Arguments	**`time_marker`** (same as tone)

Examples	`noise (400)`
	a white noise from 0 to 400 ms
	`noise ([200 500])`
	a white noise from 200 to 700 ms

gnoise	Same as noise but Gaussian-distributed noise instead of uniformly distributed.
Syntax	**`gnoise (time_marker)`**

wave	Read a wav file
Syntax	**`wave (wave_name)`**
Arguments	`wave_name:` name of wave file. Must use double-quotation marks. (If path is not included, current folder (the same directory as the psycon.exe will be searched). * extension ".wav" can be skipped
Examples	`wave ("c:\soundData\specialnoise")` Read the signal from specialnoise.wav in specified directory.

fm	Generate a frequency-modulated tone signal
Syntax	**`fm (freq1, freq2, mod_rate, time_marker, [init_phase=0])`**
Arguments	`freq1, freq2:` the boundary frequencies in Hz `mod_rate:` how many times the frequency will swing between freq1 and freq2. Zero indicates no swing, but just linear transition from freq1 to freq2. **`time_marker`**`:` (same as tone) `init_phase:` The initial modulation phase between 0 and 1. (0 means it begins with freq1 and 1 means it begins with freq2)
Examples	`fm(500, 700, 4, 2000)` a 2 second FM tone between 500 and 700 Hz at 4 Hz
Note	**If mod_rate is set as 0, this function generates a frequency-glide tone from freq1 to freq2.**

silence	Generate a silence signal
Syntax	**`silence (time_marker)`**
Arguments	**`time_marker`**: (same as tone)

Examples	`silence (400)`
	silence from 0 to 400 ms
	`silence ([200 500])`
	silence from 200 to 700 ms

Note	This function is necessary to generate a signal with no content in some experiments

step	Generate a dc signal
Syntax	**`step (time_marker)`**
Note	This is equivalent to 1+silence(time_marker); necessary for windowing (see below examples)

4. Signal modification functions

Modifications of a signal generally has the following format:
Function_name(target signal, additional arguments)

rms	Scale the signal to make the rms of specified value

Syntax	**`rms(signal)`**
	`rms(signal, target_level_db)`
	`rms(signal, ref_signal, signal_ref_ratio_db)`

Arguments	`signal:` any valid signal **`target`**`_level_db:` The target rms level to scale the signal. This value is referenced by the maximum calibrated level (the full scale of the equipment) `ref_signal:` reference signal `signal_ref_ratio_db:` relative level with regard to reference signal

Examples	`rms(wave(“voice1”))`
	Obtain the rms of voice1.wav (in relative dB to 0 dB max)
	`rms(wave(“voice1”), 80)`
	Make the signal level to be 80 dB rrns, assuming that the max calibrated scale is greater than 80 dB (if not this will issue an error).

	`rms(wave(“voice1”),MAXCAL-10)`
	Make the signal level to be 10 dB rrns down from the full max calibrated level.
	`rms(noise(1000), wave(“ref”))`
	Scale the noise so that its rms is the same as `ref.wav`
	`rms(noise(1000), wave(“ref”), 10)`
	Make the noise rms 10 dB higher than `ref.wav`
	`rms(noise(1000), wave(“ref”), -5)`
	Make the noise rms 5 dB lower than `ref.wav`

lpf **hpf** **bpf** **bsf**	IIR filtering. Lowpass, highpass, bandpass, and bandstop filtering, respectively. lpf and hpf require at least 2 arguments (signal and edge frequency) and bpf and bsf require at least 3 arguments (signal and two edge frequencies). The rest arguments are optional and available for tweaking detailed filter settings.

Syntax	**`lpf(signal, freq, [order=8], [kind=1], [dBpass=.5],[dBstop=-40])`**
	`hpf(signal, freq, [order=8], [kind=1], [dBpass=.5],[dBstop=-40])`
	`bpf(signal, freq1, freq2, [order=8], [kind=1], [dBpass=.5], [dBstop=-40])`
	`bsf(signal, freq1, freq2, [order=8], [kind=1], [dBpass=.5], [dBstop=-40])`

Arguments	`order:` Filter order. Default=8. `kind:` 1 for Butterworth (default), 2 for Chebyshev and 3 for Elliptic `dBpass:` Passband ripple in dB (default = 0.5 dB). `dBstop:` Stopband attenutation in dB (default = -40 dB).

Examples	`lpf(noise(1000), 2000)`
	Lowpass noise with a cutoff frequency of 2000 Hz (8th order Butterworth)
	`hpf(noise(1000), 3000, 6, 2)`
	Highpass noise with a cut of 3000 Hz, 6th order Chebyshev
	`bpf(noise(1000), 1900, 2000, 6, 3, 1, -80)`
	bandpass noise between 1900 and 2000 Hz, 6th order Elliptic with 1dB ripple and -60 dB attenuation

note	Be sure to check the IIR filter’s stability. Note that some filtering specs (such as very narrow bandpass filtering) will be difficult to obtain stable filter coefficient.

filt	Filter the signal with the specified coefficients

Syntax	**`filt(signal, num_array, den_array)`**
	`filt(signal, mat_file, matlab_filter_variable)`

Arguments	`num_array, den_array:` Numerator or denominator coefficient arrays. Must use a bracket or tag (see how to use tag). `mat_file:` The matlab file (*.mat) that contains the variable specified in the 3rd argument. Must use double-quotation marks. This could include a path. Default extension is .mat. `matlab_filter_variable:` The matlab variable of filter coefficient generated by sptool. Must use double-quotation marks.
Examples	`filt(noise(1000), [.2 -0.2 .001], [1 -.2 .02])` Filter the white noise with these coefficient arrays (2nd order IIR)

	`filt(noise(1000), “filters”, “lp1500”)` Filter the noise with the coefficients available in matlab variable lp1500 stored in filters.mat

am	Amplitude-Modulate the signal
Syntax	**`am(signal, mod_rate, [mod_depth=1], [init_phase=0])`**
Arguments	`mod_rate:` AM modulation rate in Hz
	`mod_depth:` Modulation depth between 0 (no mod) and 1 (full mod), If skipped, full modulation is assumed.
	`init_phase:` The initial modulation phase between 0 and 1.
Examples	`am(noise(1000), 8, 0.5)`
	AM of a white noise at 8 Hz with a depth of 0.5

stereo	Make a stereo signal
Syntax	**`stereo(signal1, signal2)`**
Arguments	signal1 and signal2 are signals in the left and right channel, respectively.

ramp	Make the beginning and ending points smooth (to avoid spectrum splatter)
Syntax	**`ramp(signal, ramping_time)`**
Arguments	`ramping_time:` The time in ms for ramping up (at the beginning) and down (at the end)
Examples	`ramp(tone(1500,500), 10)`
	Make the tone with 10-ms beginning and ending ramping.
Note	This uses the cosine-square function for smoothing.

5. Other functions for signal generation/psychoacoustic procedure

dur	Get the duration of signal in ms
Syntax	**`dur(signal)`**
Note	This is used to make an argument of different expression in a complex signal

length	Get the length of signal in sample size
Syntax	**`length(signal)`**
Note	This is used to make an argument of different expression in a complex signal

rand	Generates a double-precision random number
Syntax	**`rand(max_val)`**
Note	Double-precision random number between 0 and max_val (with a uniform distribution)

irand	Generates an integer random number
Syntax	**`irand(max_val)`**
Note	Integer random number between 1 and max_val (with a uniform distribution)

randperm	Generates a randomly permuted array from 1 to val
Syntax	**`randperm(val)`**
Note	An integer array from 1 to val and its order is randomly permuted is generated.

file Syntax	Reads a text file that contains an array of numeric values (deliminated by CR/LF) **`file (text_file_name)`**

6. Math functions

sin	Sine
cos	Cosine
log	Natural log
log10	10-base log
abs	Absolute value
exp	Exponential
^	a^b (a raised by b)
sqrt	Sqrt root
mod	Modulo

7. Variable definitions

As the signal construction becomes complex, it is convenient to define a certain signals(s) or value(s) ahead of time and use them later. The Variable Definition ("Tag Definition" in older versions) edit box serves this purpose. Each line in this edit box may have the syntax of (variable) = (expression), so the (variable) can be referred in the main signal construction. For example,

```
a = 0.5*tone(440,300)
b = 0.25*tone(880,300)
```

can be defined in the Variable Definition box and simply a + b can be used in the signal box. Or

```
whitenoise500 = ramp(noise(500),10)
```

can be defined above and a more complex syntax can be used in the signal construction such as:

```
am(bpf(whitenoise500, 2000, 4000), 8, db(-25))
```

In a verbal expression, the above signal is "a bandpass noise with the duration of 500 ms and gradual ramping for 10 ms applied at the beginning and ending, with cutoff frequencies of 2000 and 4000 Hz, amplitude-modulated by 8 Hz with the depth of -25 dB."

8. Looping

A single-line looping can be made with the function sigma:

sigma	Accumulate the signal with the given formula
Syntax	**`sigma(`*`var=range`*`, `*`formula_of_var`*`)`**
Arguments	`var:` variable to loop ***`range`***: same as MATLAB format using colon(:). If specified by two values, integer spacing, if specified by three values, the second value is the step size. For an arbitrary numbers, use a bracket [ ].

Examples	`sigma (i=0:4, tone(2000, [200*i 100]) )`
	100-ms 2000Hz tone on/off with a period of 0.2 second repeating 5 times
	`sigma (i=0:4, sigma(k=0:2, tone(2000, [400*(i+k/5) 50]) ) )`
	50-ms 2000Hz tone pips repeating three times with the period of 80 ms then the pattern repeating with the period of 400 ms for 5 times (beep beep beep, beep beep beep, beep beep beep, beep beep beep, beep beep beep)

Looping expressions can also be used across multiple lines in the variable definition box, using one of the following: if, while, and for.

```
if (condition)

end

while (condition)

end

for (index_variable_definition)

end
```

For example, the following example demonstrates how a noise-vocoder (so-called cochlear implant simulation) can be defined in variable definition box (this generates y):

```
freq = [500 1500 4000]
x = wave(“test.wav”)
y=[]              ← initialization of output signal with null
for i=1:4
     if i==1
          signalband = lpf (x,freq(i))
     else if i==4
          signalband = hpf (x,freq(i))
     else
          signalband = bpf (x,freq(i),freq(i+1))
     end
     envelope = lpf(abs(signalband),200) ← low-pass filtering full-
wave rectified signal
     noisemod = envelope * noise(duration(envelope))
     if i==1
          signalband2 = lpf (noisemod,freq(i))
     else if i==4
          signalband2 = hpf (noisemod,freq(i))
     else
          signalband2 = bpf (noisemod,freq(i),freq(i+1))
     end
     signalband2 = rms(signalband2, signalband) ← scale
signalband2 to maintain the same rms energy before noise modulation
     y = y + signalband2
end
```

The function cisim does the processing described above:

cisim	Noise-vocoder (cochlear implant simulation)
Syntax	**cisim(signal, number_of_bands)**

9. Non-signal array

An array may be defined using the bracket, as used in `filt(noise(1000), [.2 -0.2 .001], [1 -.2 .02])`. It might also be useful to use along with the use of a tag. Another example: to play a musical melody (C-E-G) followed by a triad chord (a "C" chord)

Tag definition:
```
mel=[0 4 7]/12
```

Signal:
```
sigma ( i=0:2, ramp(tone(500*2^mel[i], [400*i 250]),10) ) +
sigma ( i=0:2, .25*ramp(tone(500*2^mel[i], [1200 500]),10)
)
```

The numbers inside of the bracket may be either an array, a sequence (following the notation of MATLAB) or the combination of both. For example,
`x=[0 2 4 6]` can be written as `x=[0:2:6]`
`x=[4 1:3:5 0 7]`

An array definition may contain other tag defined earlier, for example,

```
a = [4 5 6]
b = [a 10]      → same as [4 5 6 10]
b = [2*a 10] → same as [8 10 12 12]
```

`x=[sin(2*3.141592*500*(0:11025)/22050)]` ← This low-level definition is to be used instead of the `tone` function. But not yet supported in version 1.1

10. Indexing with tags defined

The use of tags allows extraction of data samples or time-scaling from defined tags.

i) bracket [ ] ---originally to be used for a non-signal array (not a signal is OK, too) Integers must be used inside of brackets and they indicate the indices of the array defined (see the example `mel` above). (note: the index is zero-based, i.e., the first element is 0, not 1). For example, if x is defined as a sequence with the step of 2, from 2 to 20 (i.e., `x=[0:2:20]`

`x[0:2:10]` → same as `x[0 2 4 6 8 10]` → `[0 4 8 12 16 20]`

ii) parenthesis ( ) – to be used for a signal (yet to be implemented as of version 1.50)

The values inside of the parenthesis, which can be *any* value, are in milliseconds. The use of parenthesis is in one of the following formats (assuming that x has been defined)

`x(t1:t2)`	x extracted from t1 to t2.
`x(->t)`	x delayed by t (zero padded until t)
`x(<-t)`	x fast-forward by t (x values now before 0 truncated)
`x(^p)`	time-scale slowed by factor of p (pitch lowered)
`x(vp)`	time-scale sped by factor of p (pitch increased)

11.Integrative examples

11.1. AM-FM signal

```
.5*am(fm(500,1000,6,1000),3, db(-15))
```

1 second 500/1000Hz FM tone at 6 Hz, also AM at 3 Hz with a modulation depth of –15dB

```
ramp(.5*am(fm(500,1000,6,1000),3, db(-15)),10)
```

10-ms ramped version of above signal

11.2. Speech plus filtered noise with a SNR of 5 dB

```
wave(“voice”) + rms( filt(noise(dur(wave(“voice”))),
“filters”, “lp1”), wave(“voice”), -5)
```

Using a tag,

```
A = wave(“voice”)
A + rms(filt(noise(dur(A))), “filters”, “lp1”), A, -5)
```

Make the rms of total signal 80dB (assuming that the max cal level is greater than 80 dB)

```
rms(A + rms(filt(noise(dur(A))), “filters”, “lp1”), A, -5),
80)
```